Proposal for OSG Application Software Installation Service (OASIS)

Bockelman, Brian
Caballero, Jose
De Stefano, John
Hover, John

December 3, 2012

1 Introduction

This document outlines the motivation and requirements for the OSG Application Software Installation Service (OASIS). OASIS is a value-added service the OSG may offer to smaller VOs to assist with the distribution of VO application software. This service would not be mandatory, but we believe would significantly lower the barrier in becoming an OSG VO.

The basic concept is the OSG will offer an access mechanism to the VOs software managers. The managers will be able to login and install their application software in a central location. Once installed in the central location, the OSG is responsible for distributing the VO software to as many sites as possible. In exchange for using the OSG-provided service, the VO will have to abide by a certain set of restrictions or policies such as total space used and/or number of files.

The requirements outlined in this document cover what we believe is needed for the first version of OASIS; we highlight requirements that may be done in the future (implicitly, any such requirements are not part of the first version).

2 Background

The OSG has always encouraged the concept of software portability: a users scientific application should be able to run in as many operating system environments as possible. This is typically accomplished by compiling the software into a single static binary, or distributing all the dependencies in a tarball downloaded by each job. However, the concept of portability runs against the

current Linux software distribution philosophy, and becomes increasingly difficult to achieve as the size of a scientists software stack increases. At this point, portability is a barrier adoption of the OSG and it is not pragmatic to provide software packaging assistance to every OSG VO.

Accordingly, it is necessary to provide a mechanism for OSG VOs to pre-install software at the sites where they run.

Since its first release, the OSG CE has provided a directory to VOs, referred to as "OSG APP" (for the job environment variable, $OSG_APP, that points to its runtime location), that can be used for application software installations. The VO assumes it can create a sub-directory that it owns, install its software into the sub-directory, and have the directory shared on the worker nodes for a site. The OSG provides guidelines for the size and UNIX permissions of the $OSG_APP directory.

For example, CMS installs its applications inside $OSG_APP/cmssoft. The initial installation takes about 10GB of disk space, and each subsequent install can take another 1GB. Only CMS has the ability to write into $OSG_APP/cmssoft.

The $OSG_APP model has shown to have a few issues:

- It provides no enforcement or allocation mechanism. A single *badly-behaved* VO can utilize all the space available, denying service to other *good* VOs. When this happens, the local site has to make decisions about how to clean up the shared space. In other words, no quota policies can be enforced.
  - The site likely has an internal prioritization for one or two VOs. An ATLAS site will give ATLAS priority in $OSG_APP, but likely does not care about the rest of the OSG VOs.
- It may be implemented inconsistently across sites. Some sites prefer VOs do the installation from the worker nodes; others have $OSG_APP read-only from the worker nodes. VOs typically have to learn the idiosyncrasies site-by-site, and there is little sharing between VOs. Also, when a new site comes to OSG, the whole software package has to be installed for all VOs the site intends to support.
- The site needs to create and maintain the special UNIX accounts.
- Common software are often installed in multiple places.
- Unless sites provide for dedicated WNs, or specific batch policies, installation jobs may need to wait in a queue.
- OSG provides no toolset for distributing software. Sites provide a writable directory, but VOs may want higher level concepts such as "copy this file to all accessible sites".

A new mechanism, allowing the distribution of the new software after a single installation, is desirable.

3 Architecture

The proposed architecture for OASIS is a server-client model. Installation is performed once, at a central place, and the software is distributed automatically to all sites.

The proposed architecture implies that software is installed in a single place (or a reduced number of places), and that the software will be automatically distributed to all sites. This dedicated area is otherwise immutable and will not accept any other type of input, since the installation procedure requires authentication and authorization in order to run jobs that update or modify the software store.

The software and data deployed at the server is automatically distributed to client sites. It would be desirable for the sites to run some type of replica (or cache) service, to prevent all WNs to read directly from the server. Clients then get the content from this site replica, preferably throught one or more Squid caches [1].

Clients should trust content based on X509.

In principle, it would desirable to have an unique OASIS server, at a single place, for all VOs. The proposal is the Grid Operation Center (GOC) to host this OASIS server.

In case there are more than one OASIS server (because each VO wants to run its own one, for example) then the OASIS clients will need to query periodically an external service to retrieve the update list of servers. This mechanism may introduce unnecesary security implications, makes the OASIS client more complex, and adds extra requirements. For example, the list of server sites (including both multipurpose ones and VO specific) should have to be published. If this approach is needed, the proposal is the GOC to include this information as part of OIM. The OASIS client downloads that information periodically and restart when new configuration arrives. Recommended tool to manage this automatic configuration update is Puppet [2].

For this server client architecture, OASIS has to rely on some existing file distribution. OASIS will not implement its own dedicated distributed filesystem technology. The proposal for such underlying technology is CVMFS [3].

3.1 Server

3.1.1 Login

The proposed mechanism allows login via a gatekeeper. Only credentials carrying a special VOMS attribute should be accepted. When a new VO is created, the authentication and authorization mechanism in place at that site should add the appropriate new accounts for that VO's authorized managers. Each VO manager's credentials should be mapped to a different UNIX account, so that their installation jobs do not interfere with those being performed by other VOs. This also implies that, for new VOs, the corresponding UNIX account must be created.

Optionally, if a special software area for common tools is to be shared between multiple VOs (like ROOT or GEANT for HEP communities, or BLAST for Biocomputing communities), credentials with an special VOMS attribute (shared by the VOs) maybe be needed. These credentials are mapped to a dedicated UNIX account too.

The alternative would be that all UNIX accounts have permissions to write in this common area. But in that case, when a VO is using their own service and, therefore, is not allowed to use their own area at central OASIS server, then it could not interact with this common area either.

Granting login to the server via ssh can be considered if it is required by a given VO.

3.1.2 Access to sofware server area

There are three possibilities for the installation jobs to access the final software area at the distribution service server:

- ssh access,
- the server is installed on the same host where the jobs are running,
- server and working node share a filesystem.

Recommended solution is the third one. First option is unnecesary and may introduce extra complexities. Second option does not work in case of more than one working host (for reliability reasons).

3.1.3 VO software installation and deployment

As mentioned, the login access to the central services is granted via a dedicated gatekeeper. The gatekeeper gives access to a local batch queue with a very few nodes, managed by condor [5]. The installation includes a dedicated condor job wrapper script (pointed by condor configuration variable

USER_JOB_WRAPPER). Documentation on this wrapper script can be found at [6].

This wrapper script can eventually perform a set of extra tasks, aimed basically to enforced the policies:

- Checks if there is enough space available for that VO to proceed,
- Can deny more installation tasks if there is very little free space.
- Checks how much left space is remaing after last installation. If it is 0 (or very close), it will assume the VO went over-quota.
- If the installation job completed with success, and VO did not run over-quota, performs an atomic publishing command.
- If the task did not succeed, can delete the files created during the failed installation attempt.
- Writes a logbook recording information like: identity (DN), timestamp, change in service catalog, results of the installation operations (success, failed, over-quota, etc.) This logbook can be usefull for troubleshooting, collecting usage statistics, etc.

If needed, that wrapper can check the VO the job belongs to and call a dedicated vo_wrapper with VO-specific tasks.

VO software installation and deployment operations can be split into two separate steps. That means VOs would have to prepare two type of jobs: one to install new software, and one to deploy it. In this scenario, we understand by `installation` the process to install new files on a testing server. Policies on this testing server can be more relaxed, allowing deleting files (or rolling back previous installations), re-writing files, etc. The step of `deployment` implies moving the new content into a production server. Policies here are more strict, not allowing deletion or rewriting of existing files.

Every step being performed should be reported in the logbook service. This will allow rolling back, for example, among other features.

The OASIS server package would include the necessary code to perform these deployment operations, moving files from the testing repository to the production one. This code can be the one enforcing quota policies, no deletion policies, etc.

3.1.4 OASIS daemon

The proposal for the server includes a dedicated OASIS daemon in charge of managing internal tasks (for example, running the CVMFS server publishing scripts).

There are some operations can not be performed directly via grid job. The reason is that some operations need to re-start some server services or daemons. The new service is from that point on considered a child process belonging to the grid job. Therefore, when the job finishes, the local batch system will kill all these child processes too.

The proposed solution is to have a dedicated OASIS daemon running at the server. The user jobs will leave messages to this OASIS daemon, which will address the request and perform the required operations on behalf the user. The communication between the user job and the OASIS daemon can be done, for example, via messages in a database.

3.2 Client

The OASIS client will be distributed also as RPM. It will require the CVMFS client to be already deployed as a dependency. The OASIS client RPM will place the dedicated configuration file in the right directory (/etc/cvmfs/domain.d/), will create if does not exist /etc/cvmfs/default.local, or will create a backup copy otherwise. The RPM will also copy the public key into /etc/cvmfs/keys/ directory

Sites will need to deploy the CVMFS client on every WN. In the future, CVMFS client will be compatible with NFS, allowing having a single CVMFS client and distributing the files over the network. But that feature is not yet in place.

VO software should be then distributed at the sites under directory

```
/oasis/<myvo>/
```

However, given there may be VOs not interested on using OASIS, it would be desirable to keep $OSG_APP on the worker nodes.

On the other hand, even VOs migrating to OASIS service may have, for historical reasons, the string $OSG_APP hardcoded in their jobs software. In that case, it would make sense, to avoid forcing them to change their code, to create links at the sites from old $OSG_APP area to the new OASIS one:

```
$OSG_APP/<myvo> -> /oasis/<myvo>/
```

4 Policies

4.1 Space quota

Each VO should be guaranteed a given amount of space when using the central service provided by OSG. The amount of space granted to each VO is set by

the OSG Execute Team. If a VO needs additional space, it should consider the deployment of its own, separate repository service. In that case, that VO should not be allowed to have space at the central service (but would still be allowed to manage the common area).

Enforcing the limit on space quota can be achieved at the level of filesystem [4].

4.2 Content

Only software with public licenses can be deployed. The VOs must agree to permit OSG the right to inspect the content of the software to ensure that their software and data are in compliance with relevant rules and laws.

A dedicated area could be used to distribute the OSG WN middleware too.

4.3 Recommended procedures

In order to avoid inconsistencies between different replicas of the same file, it would be desirable to prevent the deletion and/or rewriting of files at the code source site.

If file deletion and re-writing are forbidden, all files must have a clear, well defined version number to distinguish one version from another.

5 OASIS distribution

OSG will publish the packages (RPMs, configuration files, etc.) for the server and, if needed, for the client.

There is a question to be solved. If the underlying technology used by OASIS is CVMFS, how to proceed in case the site already have the CVMFS client running, pointing to a different CVMFS server (for example, at CERN)? Using the existing package implies OASIS can not stick with a frozen release whne new features are not desired. In case of distributing a dedicated CVMFS client package for OASIS, namespaces conflicts may occur.

Appendix A Experience

Appendix A.1 Motivation

In addition to gaining expertise, and to simply testing an existing set of technologies, one of the goals of this exercise was to determine how flexible the CVMFS server installation could be: OSG desires to use the software as part of its planned architecture to distribute software repositories, but to obfuscate the underlying technology and unify its pieces under the OASIS project banner. For example, on the CVMFS server, the default mount point for software storage and distribution is `/cvmfs`; OSG would like to change this to `/oasis`. This was suspected to be problematic, as no documentation existed to support such a mount rename or relocation, and software configuration knobs to this effect are sparse and also undocumented.

Appendix A.2 Infrastructure

Two virtual RHEL 6 nodes: one for the CMVFS client and one for the server. A third node (RHEL 5 VM) was later added.

Appendix A.3 General Installation and Troubleshooting

The client installation went smoothly, after working through a few bugs with packages and keys, but the server did not: a CVMFS server RPM is available for RHEL 6, but it contains hidden dependencies for CVMFS RPMs that are currently only available for EL 5: RedirFS, and a CVMFS kernel filter that depends on RedirFS. RedirFS was installed from official source, which worked, but compiling and installing the CVMFS RedirFS kernel filter failed due to a parameter compatibility issue. The RedirFS and kernel filter modules were then both built from the CVMFS source, but the kernel filter module still failed to start with the same error.

After multiple troubleshooting sessions and conversations with the CVMFS developers, a decision to revert to RHEL 5 for the server operating system was made. The previous RHEL 6 VM originally allocated for the server component was reconfigured as a CVMFS replica server. As with the CVMFS client on RHEL 6 before, after making a few package setting adjustments, the installation of a CVMFS server on RHEL 5 was relatively simple; however, as expected, the customization of the basic server mount point brought with it many complications.

As a result of these findings, the customizations made to the current server were reverted, and the default settings and locations were used for testing server functions. To do so, the custom mount had to be destroyed, and two new ones created to accommodate the default CVMFS server mount locations:

- `/cvmfs` for source sofware storage,

- `/srv` for CVMFS repositore storage.

It was also necessary to install a Squid repository on the client system (ideally, this would be installed on its own host) that could interact with the replica server as a cache to the client.

Appendix A.4 Installation details

The `iptables` firewall rules on all involved hosts must be modified in order to enable traffic between the server, replica, cache, and client.

Appendix A.4.1 RHEL 5 server install

1. Install `cvmfs-release` from remote URL.

   The `cvmfs-release` package is meant to automate installation of the CVMFS yum repository file and public signing key; however, the package is missing its own signing key and works only for Scientific Linux, not RHEL or other Enterprise Linux clones. As such, to install the package after downloading, override the default yum GPG key check:

   ```
   yum --nogpgcheck install cvmfs-release-*.rpm
   ```

   Afterward, replace the base URL in the yum repository file (`/etc/yum.repos.d/cernvm.repo`) with the functioning URL.

2. Ensure adequate space in two directories/partitions:

   - `/cvmfs`
   - `/srv`

3. Install the yum packages necessary for installing the CVMFS server:

   ```
   yum install cvmfs-server cvmfs-keys httpd redirfs \
   kmod-redirfs cvmfsflt kmod-cvmfsflt
   ```

4. Test load the RedirFS and CVMFS kernel filter modules (no returned output indicates a successful test):

   ```
   modprobe redirfs && modprobe cvmfsflt
   ```

5. After installation, the file `/etc/cvmfs/replica.conf`, which is an artifact of a package split, should be deleted.

Appendix A.4.2 CVMFS server and source repository configuration

1. Copy (`/etc/cvmfs/server.conf`) to a local override file (`/etc/cvmfs/server.local`) for customization.
   Note: this was done for testing purposes, which revealed that the desired modifications to source and shadow directories created a problematic instance. As a result, these customizations were rolled back to their defaults in the current instance.

2. Create a new CVMFS server file system and repository structure:
   `cvmfs_server mkfs {repository}`

3. Touch the otherwise unused master replica file (undocumented bug fix):
   `touch /srv/cvmfs/{repository}/pub/catalogs/.cvmfs_master_replica`

4. Synchronize data from the software source directory to the mirror tree:
   `cvmfs-sync`

5. Publish the repository:
   `cvmfs_server publish`

6. Copy the public signing key generated by the CVMFS server for the repository from the server to both the replica and the client:
   `/etc/cvmfs/keys/{repository}.pub`

7. Start the CVMFS server daemon to monitor future updates:
   `service cvmfsd start`

8. Script/cron regular repository syncing and publishing:
   `/etc/cvmfs/cvmfs_publish.sh`

```
#!/bin/#!/bin/bash
chown -R cvmfs.cvmfs /cvmfs/testrepo.racf.bnl.gov
cvmfs-sync >>/var/log/cvmfs/update.log
cvmfs_server publish >>/var/log/cvmfs/update.log
```

```
/etc/cron.d/cvmfs_publish
```

```
*/20 * * * * root /etc/cvmfs/cvmfs_publish.sh > /dev/null 2>&1
0 0 * * 0 root /usr/bin/cvmfs_server resign >>
/var/log/cvmfs/update.log 2>&1
```

Appendix A.4.3 RHEL 6 replica installation

1. Follow Step 1 of the server installation section to install the CVMFS repository file and public key.

2. Install the replica and Apache packages:
   `yum install cvmfs-replica cvmfs-keys httpd`

3. Add an entry to the CVMFS repository mirror file (`/etc/cvmfs/replica.repositories`) to point to the CVMFS server URL and repository path, signing key, and local path:

   ```
   testrepo.racf.bnl.gov|http://grid24.usatlas.bnl.gov:80
   /cvmfs/testrepo.racf.bnl.gov|/etc/cvmfs/keys/testrepo.
   racf.bnl.gov.pub|/var/cvmfs/testrepo.racf.bnl.gov/pub|32|10|3
   ```

4. Configure Apache to serve the repository replica:

   - `/etc/httpd/conf.d/cvmfs-replica.conf`

     ```
     RewriteEngine on
     # /opt/<experiment> is forced to be lower case:
     RewriteMap toLower int:tolower
     # Automatically point to the catalogs:
     RewriteRule ^/opt/([A-Za-z0-9\-\.]+)/(.*)$
     /opt/${toLower:$1}/pub/catalogs/$2 [PT]
     # Translation URL to real pathname:
     Alias /opt /var/cvmfs

     # Close access to non-pub directories
     <DirectoryMatch /var/cvmfs/[A-Za-z0-9\-\.]+/(shadow|ctrl)>
     Order allow,deny
     Deny from all
     </DirectoryMatch>

     <Directory "/var/cvmfs">
     Options Indexes -MultiViews FollowSymLinks
     AllowOverride All
     Order allow,deny
     Allow from all
     IndexOptions +SuppressDescription +FoldersFirst +VersionSort
     +SuppressIcon
     EnableMMAP Off

     EnableSendFile Off
     AddType application/x-cvmfs .cvmfschecksum .cvmfscatalog .x509 \
     .cvmfspublished .cvmfswhitelist
     FileETag INode MTime Size
     ExpiresActive On
     ExpiresDefault "access plus 3 days"
     ```

```
ExpiresByType text/html "access plus 5 minutes"
ExpiresByType application/x-cvmfs "access plus 15 minutes"
</Directory>
```

5. Restart Apache:
   `service httpd restart`
6. Add a cron to update the local repository replica snapshot from the remote source:

```
20 * * * * root /usr/bin/cvmfs_snapshot > /dev/null 2>&1
```

Appendix A.4.4 RHEL 6 cache installation

The ATLAS Frontier Squid package was installed to create a cache on the same machine as the test client. After installing the software via RPM , a single modification to the default settings was necessary to enable Squid to serve CVMFS content on the local network: in `/etc/squid/customize.sh`, edit the local access control list definition to include the local machine itself , and the network on which the replica and source server reside:
`setoption("acl NET_LOCAL src", "127.0.0.1 130.199.0.0/16")`

As a precaution, the default cache directory size was also increased from 10 GB to 40 GB:
`setoptionparameter("cache_dir", 3, "40000")`

The `frontier-squid` service was then restarted in order to enact these customizations.

Appendix A.4.5 RHEL 6 client installation

1. Follow Step 1 of the server installation section to install the CVMFS repository file and public key.
2. Install the client packages:
   `yum install cvmfs cvmfs-init-scripts cvmfs-keys`
3. Run the initial client configuration script:
   `cvmfs_config setup`
4. Create the following configuration files and example content to point to the new repository via the replica server (not the software source server):
   - `/etc/cvmfs/default.local`

```
CVMFS_REPOSITORIES=testrepo.racf.bnl.gov
CVMFS_HTTP_PROXY="http://localhost:3128"
```

- `/etc/cvmfs/domain.d/{domain-of-repository}.conf`

```
CVMFS_SERVER_URL=${CERNVM_SERVER_URL:="http://grid22.racf.bnl.gov/opt/@org@"}
CVMFS_PUBLIC_KEY=/etc/cvmfs/keys/cern.ch.pub: \
/etc/cvmfs/keys/testrepo.racf.bnl.gov.pub: \
/etc/cvmfs/keys/cernit2.cern.ch.pub:/etc/cvmfs/keys/cern-it3.cern.ch.pub
```

- `/etc/cvmfs/config.d/{repository}.conf`

```
CVMFS_SERVER_URL=http://grid22.racf.bnl.gov/opt/testrepo.racf.bnl.gov
```

5. Restart CVMFS and autofs:
   `service cvmfs restartautofs`
   `service cvmfs restartclean`

6. Test the new client repository mount point:
   `ls la /cvmfs/{repository}`

Appendix A.4.6 Confirmed Issues

According to developers, the current CVMFS server tools do not support main mount point renaming or customization of any kind; they were written exclusively to support the operations of creating and maintaining repositories in a controlled environment at CERN. A rewrite of the server mechanism is underway and may be available for test soon.

Installing the CVMFS Release package cleanly requires a public signing key that is not included with the package itself. The package creates a yum repository file that is not platform-agnostic and requires a correction of the base URL in order to function.

All recommended CMVFS package fixes and improvements have been forwarded to the developers.

Appendix B Hardware requirements for CVMFS

The following comments describe the deployment of a CVMFS server at the University of Wisconsin.

High availability can be achieved well enough by mirroring the repository. cvmfs-pull can be used to do that every 3 hours and simply rely on the cvmfs client to fail over, which mostly works, although a bug has been observerd in client failover:
https://savannah.cern.ch/support/?131721

One could work around that by instead using reverse proxies in front of the CVMFS server to handle failover.

To ensure against loss of data, one possible architecture is:

- disk 1: contains writeable copy of the repository maintained by OSG VOs
- disk 2: contains the shadow tree, maintained by running rsync against writeable copy
- disk 3: contains the web files published by cvmfs

If disk 1 and 2 were both lost, data can be recovered from disk 3 (or the mirror), but ownership of files would have to be restored by some other means, because everything in the published repository is owned by the cvmfs user, whereas the file trees in disk 1 and 2 are owned by the various software maintainers (VO accounts) that manage the files in OSG_APP.

Since the web proxies do most of the work of serving data to clients, a cvmfs server could mostly be just occupied with running the periodic rsync and cvmfs publication. For a O(200G) file tree, this takes about half an hour on hardware like this:

```
2x Dual-Core AMD Opteron(tm) Processor 2212
16GB RAM
3x 1TB 7200rpm disks
```

References

[1] http://www.squid-cache.org/

[2] http://puppetlabs.com/

[3] http://cernvm.cern.ch/portal/filesystem/

[4] http://www.yolinux.com/TUTORIALS/LinuxTutorialQuotas.html

[5] http://research.cs.wisc.edu/condor/

[6] http://research.cs.wisc.edu/condor/manual/v7.6/3_3Configuration.html#19134